# The Danish National Energy Data Lake

*Requirements, Technical Architecture, and Tool Selection*

Ben Hamadou, Hamdi; Pedersen, Torben Bach; Thomsen, Christian

*Published in:*
2020 IEEE International Conference on Big Data (IEEE BigData 2020)

*DOI (link to publication from Publisher):*
10.1109/BigData50022.2020.9378368

*Publication date:*
2020

*Document Version*
Accepted author manuscript, peer reviewed version

# The Danish National Energy Data Lake: Requirements, Technical Architecture, and Tool Selection

Hamdi Ben Hamadou
*Department of Computer Science*
*Aalborg University*
Aalborg, Denmark
hamdibh@cs.aau.dk

Torben Bach Pedersen
*Department of Computer Science*
*Aalborg University*
Aalborg, Denmark
tbp@cs.aau.dk

Christian Thomsen
*Department of Computer Science*
*Aalborg University*
Aalborg, Denmark
chr@cs.aau.dk

***Abstract*—Renewable Energy Sources such as wind and solar do not emit CO2 but their production vary considerably depending on time and weather. Thus, it is important to use the *flexibility* in device loads to shift energy consumption to follow the production. For example, an Electrical Vehicle (EV) can be charged very flexibly between arriving home at 5PM and leaving again at 7AM. Utilizing all available energy flexibility requires applying machine learning and AI on massive amounts of Big Data from many different actors and devices, ranging from private consumers, over companies, to energy network operators, and using this to create digital solutions to enable and exploit flexibility. The project *Flexible Energy Denmark (FED)* is building the foundation for this for the entire Danish society. Specifically, FED collects data from a number of *Living Labs (LLs)* in representative real-life physical environments. The data is stored in the Danish National Energy Data, called FED Data Lake (FEDDL) to enable efficient and advanced analysis. FEDDL is built using only open source tools which can run both on-premise and in cloud settings. In this paper, we describe the requirements for FEDDL based on a representative LL case study, present its technical architecture, and provide a comparison of relevant tools along with the arguments for which ones we selected.**

***Index Terms*—Data Lake, Energy Data, Living Labs, Data Ingestion, Data Governance, Data Security, Open Source, GDPR**

## I. Introduction

To reduce the CO2 emissions, energy from Renewable Energy Sources (RES) should be used as much as possible. However, their production fluctuates as, e.g., wind turbines only produce energy when there is wind and solar panels only produce energy when there is daylight. To enable the Green Transition to a low-carbon society based on intermittent RES, the FED[1] project brings together different Danish *research institutions*, companies from the energy technology sector (referred to as *EnergyTech companies*), and owners of real-life physical environments where energy usage (and other relevant measures, e.g., indoor climate) can be monitored and experimented with. The latter are called *Living Labs* (LLs) and represent different parts of the Danish society. For example, there is a LL representing a Distribution System Operator (DSO) which operates the electricity grid transporting electricity from the larger transmission system to end-users as well as a LL representing housing associations. In every LL, the owner works together with relevant research institutions and EnergyTech companies to identify and exploit *flexibility* where energy consumption can be shifted to follow the available RES production. For example, charging of an Electrical Vehicle can happen very flexibly between arriving home at 5PM and leaving again at 7AM. Another example is to *pre-heat* an apartment within given comfort limits to balance user comfort with using available green energy.

To enable scalable and effective use of all flexibility, it is important to capture flexibility in a uniform and powerful format. In FED, we use the FlexOffer (FO) format developed in a range of large EU and Danish projects [1]. Consider the EV charging scenario from above. For this, the associated FO will have its earliest start time (for charging) at 5PM and its latest end time at 7AM. In this 14 hour *time flexibility* period, we can in each 1 hour time slice adjust the energy between 0 and 4KWh, giving an *energy flexibility* of 4 KWh. Over the whole period, the EV must be charged between 30 and 60 KWh. All these constraints and the grid location are encoded in the FO, which can then, along with millions of other FOs, be aggregated, traded, and scheduled to balance RES production with consumption as well as respect grid capacity constraints.

FED is data-driven and collects large amounts of heterogeneous data including consumption data from meters, production data, sensor data, weather observations, etc. Utilization and identification of all available energy flexibility requires applying machine learning and AI on these massive amounts of Big Data. To facilitate this, FED establishes the Danish National Energy Data Lake, called FED Data Lake (FEDDL). For this Data Lake, the typical challenges with Big Data such as Variety, Velocity, and Volume of the data have to be dealt with. Due to the nature of the project and its data, there are, however, also novel non-typical challenges, e.g, with respect to compliance with EU's data protection laws (GDPR) to ensure privcay. Further, it is a requirement that only open source

This work was supported by Innovation Fund Denmark 1
https://www.flexibleenergydenmark.com

software is used for the implementation and that deployment can happen both on-premise or in cloud settings. In this paper, we describe the requirements for FEDDL based on a representative LL case study, present its technical architecture, and provide a comparison of relevant tools along with the arguments for which ones we selected.

The major contributions of the paper include:

- A real-life case study of a Living Lab, the data it provides, and its challenges to be addressed
- The functional and non-functional requirements for FEDDL
- The technical architecture for FEDDL showing its layers and explaining how they work together.
- Arguments for our tool selection for the implementation of FEDDL based on a structured comparison of state-of-the-art open source tools.
- Deployment strategies meeting the requirements

The paper is organized as follows. We start by introducing a case study describing a Living Lab in Section II. Then, we specify the functional and non-functional requirements for FEDDL in Section III. Section IV provides an overview of the technical architecture for FEDDL. A structured comparison and a discussion of state-of-the-art tools required to build a Data Lake are presented in Section V. Implementation and deployment strategies are presented in Section VI. Finally, we conclude and discuss future work in Section VII.

## II. Case study: A Danish Distribution System Operator

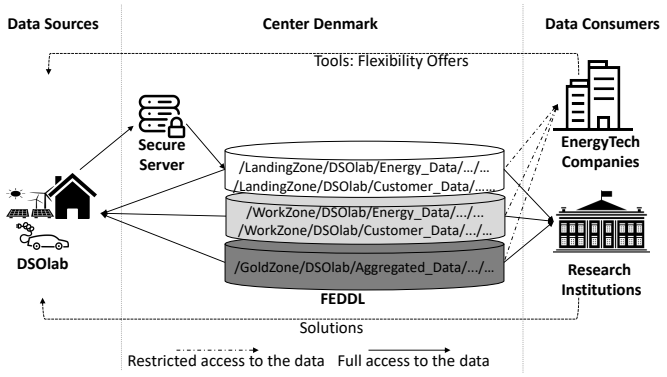


Fig. 1. Case study: (DSOlab)

In this section, we present one of FED's Living Labs referred to as "DSOlab". First, we describe DSOlab and the data that it delivers. Then, we describe the frequency and the methods used by DSOlab to deliver data. Finally, we present some of the applications using DSOlab data. Fig. 1 presents the interactions between the *Data Sources* in DSOlab, *Center Denmark*[2], and the *Data Consumers*. Center Denmark is an independent and non-profit Danish research centre aiming to unify and embed research results within the field of digitalization of energy systems and put data intelligence for potential commercial use. Center Denmark is responsible for hosting FEDDL which is a centralized repository composed of different zones, namely the *Landing Zone* where a copy of the data shared by the LLs are stored, the *Work Zone* where intermediate results are stored, and the *Gold Zone* where final results are stored so the Data Consumers can explore these new results.

The DSOlab provides data from a Danish DSO managing a distribution system for electricity at the low, and medium voltage levels (LV, MV). The electricity comes from both the national transmission system and local RES such as solar panels and wind turbines in the DSO's grid. The DSO is responsible for collecting and sharing *electricity consumption and production meter data* with the Data Consumers involved in DSOlab, and *data describing the installations* within the DSO grid area. The installations include more than 140k residential and industrial consumers, producers, and prosumers (who are both producers and consumers). The meter data contain sensitive information for the installations including the installation identifier, measurement time, consumption and production meter reading expressed in KWh on an hourly or quarter-hourly basis. In addition, the installation data contains personally identifiable information, namely the location of the installation. In the following, we refer to electricity consumption and production meter data as *energy data*, and data describing the different installations as *customer data*.

The DSO involved in DSOlab collects energy data remotely from all installations with different frequencies, i.e., on a hourly or quarter-hourly basis, based on meter type. The DSO sends the energy data of 1k installations to a dedicated secure SFTP server on a daily basis at 8 AM with data for the previous day (00 – 24). Also, DSOlab has historical energy data of all 140k installations, for at least four years. The DSOlab also provides customer data and grid data. The DSO uses Parquet file format, a column-oriented data storage format, for the historical energy consumption and customer data. Furthermore, in DSOlab energy consumption data is continuously sent on a quarter-hourly basis using Kafka real-time data streaming and stored in FEDDL. FEDDL ingests batch and streaming energy and customers data delivered byDSOlab and loads them into dedicated directories in the FEDDL landing zone. All data transfers between DSOlab, FEDDL, EnergyTech companies and Research institutions, are secured via secure channels using the HTTPS, and TLS over TCP protocols because of the sensitive aspect of energy and customers data shared by DSOlab.

DSOlab is facing challenges related to the optimization of the load on the grid during peak demand and eliminating overloads on the grid. A challenge identified by DSOlab is how to reduce the load on the electricity grid from the charging of electric vehicles by shifting electricity charging from one time period to another. Therefore, the EnergyTech companies and research institutions in FED explore DSOlab data loaded into FEDDL and develop data load-reducing tools or flexibility solutions. These novel tools and solutions using machine learning and artificial intelligence enable DSOlab

[2]https://www.centerdenmark.com

to determine the ways to reduce the load on the electricity grid during peak periods. Tools such as FlexOffers (FO) [1] capture flexibility that can be used to respond to the flexibility challenges. For instance, a FO schedules the charging cycle for the vehicle based on the analysis of historical user behaviour and prediction of future energy demand. The FO captures the constraints related to the time slot for charging the vehicle, e.g., overnight, the amount of energy needed for the charging cycle, e.g., battery capacity, and based on prediction models and demand prognoses, specific actions are proposed , e.g., shifting the time of charging the electric vehicle in order to reduce the load on the electricity grid.

The access to DSOlab data is ensured through FEDDL and it must be regulated with compliance to the GDPR for protection of personal data and privacy of EU citizens. Thus, only the data owner, e.g., the DSO involved in DSOlab, and research institutions are granted full access to all data in the different zones in FEDDL as illustrated in Fig. 1. Data Consumers, such as the EnergyTech companies, will have only restricted access where sensitive information is removed or anonymized. Furthermore, in FED, different users have very different backgrounds and skills. In addition to the possibility of downloading subsets of the data, authorized users can run SQL queries or even run custom programs directly on the data in FEDDL.

## III. FEDDL Architecture Requirements

In this section, we describe the functional and non-functional requirements for the architecture of FEDDL. The functional requirements help to define the main layers of the architecture of FEDDL and their corresponding core functionalities, e.g., inputs, outputs, data manipulation, or data flow. In the other hand, the non-functional requirements refer to the intrinsic qualities of the technical architecture of FEDDL, e.g., performance, or availability.

### A. FEDDL Functional Architecture Requirements

FEDDL functional requirements involve *storage of different file formats* generated from the different sources, e.g., streaming data of DSOlab can be stored using CSV formats, or historical data are available as *columnar files* like Parquet. Furthermore, the requirements involve ensuring *integrated access* to the different zones of data into FEDDL. The integrated access should provide *various access options* regardless of the level of expertise of the different FED partners. For instance, DSOlab, EnergyTech companies and research institutions can be provided with *full access* to the *raw data* by directly downloading files from FEDDL, or providing them with *structured access* and thus they can run SQL [2] queries directly on FEDDL data without the need to load data locally into their systems. For advanced users, e.g., researchers, FEDDL should provide *computing facilities* to perform advanced machine learning, e.g., running flexibility solutions developed using machine learning on top of DSOlab energy data. Other important functional requirement of the technical architecture of FEDDL is to *track* and *identify meta data* within all data loaded into the different directories and zones of FEDDL. In FED, DSOlab sends energy data that contain sensitive business information, i.e., energy and customer data that contain personally identifiable information, i.e., installation address. Such sensitive data are useful to drill down the flexibility offers to the street level and/or individuals, or to investigate energy consumption at both street level and the individual households. However, with regards to GDPR compliance [3], only authorized users can access to such sensitive information, i.e., researchers and data owner, the DSO involved in DSOlab. Other users, e.g., EnergyTech, will have restricted access to the data, and sensitive needs to be anonymized or aggregated such that no personal information is revealed. Thus, FEDDL should provide *fine-grained access management* to FEDDL and the different component in FEDDL technical architecture.

### B. FEDDL Non-functional Architecture Requirements

The non-functional requirements of FEDDL involve *ingestion* of a large volume of *batch* data, e.g., DSOlab historical energy and customers data for at least four years of more than 140K consumers, or the *real-time* data, e.g., DSOlab energy data streamed on a quarter-hourly basis for 1k installations. Also, FEDDL requirements involve that the deployment of the architecture can be done using *on-premises* or *cloud* infrastructures. Furthermore, FEDDL's technical architecture should be composed of open source tools deployed on *cluster* environment while ensuring *security*, *scalablity* to add more nodes, *high availability*, and the architecture should be *extensible* to easily integrate additional tools as the needs in FED can evolve.

## IV. FEDDL Architecture

FEDDL offers an integrated repository of a large volume of data stored in its natural format. In addition to the storage requirement, the FEDDL architecture should provide mechanisms to collect data from the different data sources and to offer onsite data processing capabilities for the targeted users. From the literature, it is a common practice to define several layers during the design phase of the Data Lake. For instance, in [4], the authors introduced a Data Lake composed of three layers, i.e., ingestion, maintenance, and querying. However, we notice from this architecture a lack of consideration for controlling the access management which is important while dealing with heterogeneous data sources and especially with data having sensitive information.

### A. FEDDL Overall Architecture

In this section we distinguish between FEDDL and its technical architecture. FEDDL is defined as a centralized repository where data are organized into directories divided to three zones as illustrated in Fig. 1, i.e., i) The *Landing Zone* which is the first stage where raw data are loaded into FEDDL. Thus, data are collected from the LLs, and stored without applying any data cleaning or processing. ii) The *Gold Zone* contains clean and well-structured data which are quality ensured, e.g., all authorized FEDDL users may

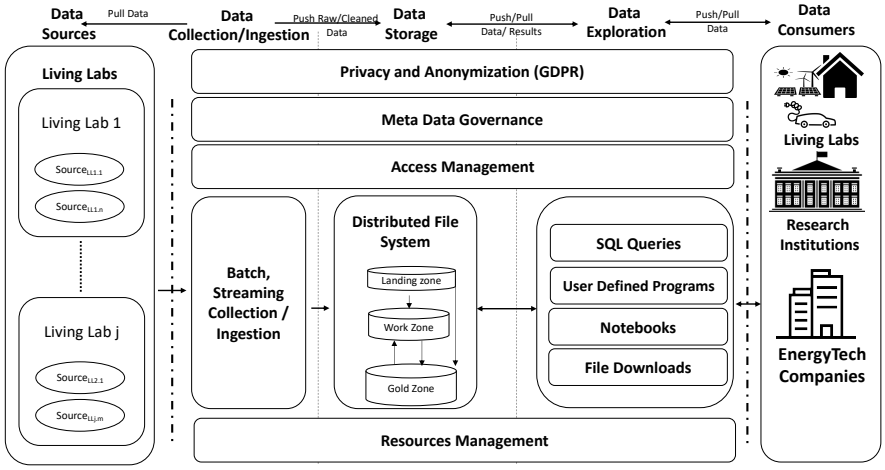


Fig. 2. FEDDL architecture

store aggregated results after applying advanced processing to the DSOlab energy data, and iii) The *Work Zone* is where FEDDL users can store intermediate data sets which are being processed, cleansed, or enriched, e.g., FO needs weather data, or energy prices in addition to the energy data to determine flexibility. Once FO determines flexibility and results are ready for use by DSOlab for instance, the results move from the *Work Zone* to the *Gold Zone* and other authorized FEDDL users can access and explore to the new materialized results. Furthermore, we notice from the literature that is a common practice to define different zones for storing data into a Data Lake [5]. Fig. 2 gives an abstract overview of the technical architecture of FEDDL composed of five separated layers, i.e., *Data Sources*, *Data Collection/Ingestion*, *Data Storage*, *Data Exploration*, and *Data Consumers*, and four cross-cutting layers, i.e., *Access Management*, *Meta Data Governance*, *Privacy and Anonymization (GDPR)* and *Resources Management*. We notice that the layers *Data Sources* and *Data Consumers* represent systems which are external to FEDDL. In other words, these two layers run outside Center Denmark. In the following, we describe the different layers and the possible interaction between them:

**Data Sources**: In FED, the *data sources* layer refers to the different participant LLs, e.g., DSOlab, sharing energy data collected from different sources, e.g., Distribution System Operators (DSOs).

**Data Collection/Ingestion**: This layer focuses on pulling batch or streaming data, e.g., energy and customer data, shared by the *Data Sources* layer, e.g., DSOlab. This layer also ensures loading the data into a dedicated landing repository corresponding to FEDDL. The data transfer process must be via secure channels since the data contain sensitive information and come from external systems.

**Data Storage**: This layer represents the central repository, i.e., FEDDL, where large volume of energy data are loaded. Thus, this layer must provide a distributed file system where data can be stored into directories to efficiently load the data while guaranteeing high availability and fault tolerance, e.g., disk failure.

**Data Exploration**: This layer allows LLs, e.g., DSOlab, researchers and EnergyTech companies to explore energy data loaded into FEDDL and to run their flexibility tools and solutions directly on FEDDL data using different solutions, e.g., SQL queries, running custom programs, or downloading subsets of the data.

**Data Consumers**: This layer represents the authorized external Data Consumers, i.e., LLs, researchers and EnergyTech, to access data stored into FEDDL, or to use Data Exploration facilities offered by the architecture of FEDDL.

In addition to the five above-described layers, FEDDL introduces four cross-cutting layers, i.e., *Access Management*, *Meta Data Governance*, *Privacy and Anonymization (GDPR)* and *Resources Management*, to respond to FEDDL function and non-functional requirements, e.g., quality, security, privacy and availability of the data. For instance, the *Meta Data Governance* layer traces the provenance of the data ingested by the *Data Collection/Ingestion* layer form *Data sources*, e.g., DSOlab. Thus, information regarding what data sets exist into FEDDL, the properties of those data sets since we are dealing with sensitive data in FED, e.g., DSOlab shares data having personal information, i.e., installations addresses. The ingestion history of the data set is necessary to keep track of all data into the different directories and zones of FEDDL. The meta data provide information related to the data quality, the value ranges, schemas, or descriptions as it was collected from the *Data Sources* layer. The *Access Management* layer is responsible of ensures a policy-based mechanism to access the different directories and zones in FEDDL, and to use the different layers composing the technical architecture of FEDDL. For instance, using access control lists help to ensure that the data are protected appropriately and only authorized users are granted access to the data with the adequate rights. For instance, EnergyTech companies can have restricted access to energy and costumer data shared by DSOlab where parts

of the data containing sensitive information are removed. Furthermore, FEDDL will run on shared infrastructures offered by Center Denmark. Thus, we introduce the *Resources Management* layer to guarantee that the different resources required to run FEDDL are up-and-running at all times and efficiently manage the usage of the offered resources while isolating FEDDL from other applications co-located at Centre Denmark. Eventually, we will integrate a custom *Privacy and Anonymization (GDPR)* layer to ensure the protection of sensitive data proper to FED. This layer is, however, not included in the first version of FEDDL (FEDDLv1) where the *Access Management* layer will be used to ensure that only data owners and authorized researchers can access data with personal information. In FEDDL, all layers run on top of the *Resources Management* layer which ensures also isolation of FEDDL services from the other services running on Centre Denmark.

### B. Data Flow in FEDDL

Fig. 2 shows the data flow between the different layers of the technical architecture of FEDDL. The *Data Collection/Integration* layer is responsible for pulling data shared by different sources involved in the *Data Sources* layer, e.g., DSOlab. The communication between FEDDL and *data sources* layer is ensured using one of the common communication protocols for ingesting batch files using SFTP for instance, or streaming data using Kafka clusters. Furthermore, communication has to be end-to-end encrypted which is very important in the FED context since data contain sensitive information, i.e., installations addresses. The *Data Exploration* layer pulls data from the *Data Storage* layer. However, only authorized FED partners involved in the *Data Consumers* layer can get access to the different services offered by the different layers of the technical architecture of FEDDL. Therefore, FEDDL ensures through the *Access Management* layer which is responsible to validate access for the *Data Storage* layer through the *Data Exploration* layer. Furthermore, the communication from the *Data consumers* layer is secured through SSH or HTTPS protocols. Thus, the *Data exploration* layer is responsible for pulling data from the different FEDDL zones, e.g., materialized results from the *Gold Zone*, and pushing them back to the *Data consumers* layer. FEDDL offers the possibility for the authorized users, e.g., EnergyTech companies, to send additional data to the *Wokr Zone* of FEDDL need for their processing. Therefore, the *Data Exploration* layer ensures pushing data to the *Storage Layer*. Another cross-layer communication is ensured between the *Access Management* layer and the *Meta Data Governance* layer where the *Access management* layer pulls data, e.g., tags, to limit access for a particular part of the data for instance.

## V. Structured Comparison of State-of-the-art Tools for Data Lake

In this section, we survey open source state-of-the-art tools, that can be run using on-premises or cloud environment, required to implement the different layers defined in the architecture of FEDDL. In the survey, we thus study tools which enable us to meet the FED requirements. Furthermore, we select only open source tools that have been actively studied in academia with significant research papers and/or widely adopted in the industrial field. All tools are under the Apache License and run in common cluster environments.

### A. Data Ingestion/Collection

The process of data collection/ingestion collects batch and streaming data shared by the LLs, e.g., DSOlab. The data are available in various file formats, e.g., Parquet, JSON, or CSV, and have be loaded into the landing zone of FEDDL using a given file format, e.g., Parquet. Furthermore, this process should be ensured via secure channels since data contain sensitive information, e.g., DSOlab shares data with sensitive information regarding the installation addresses. For this group, we consider three tools: Apache Nifi, Apache Flume, and Apache Sqoop.

**Apache Nifi** [6] is a distributed system, that migrates, monitors and manages data flows between disparate systems while supporting data routing and transformation through a graphical user interface (GUI) and command line interface. Apache Nifi was initially developed by the National Security Agency NSA, in Java.

**Apache Flume** [7, 8] Apache Flume is a distributed system, that migrates and aggregates large amounts of data like log files, events, etc., from a number of different sources to a centralized data store, e.g., HDFS. Apache Flume was developed by Cloudera in Java.

**Apache Sqoop** [9, 10] Apache Sqoop is a command line tool designed for transferring batch data between Apache Hadoop and relational databases. Apache Sqoop was developed by Cloudera in Java.

For this class of tools, we study the *Supported Communication* since data are gathered from heterogeneous sources, and partners may deliver data using different technologies, e.g., SFTP, MQTT, or Kafka. Furthermore, we consider the support of ingesting *Near-Real-time/Streaming* data, e.g., DSOlab continuously generates energy data on a quarter-hourly basis. Futhermore we study the supported *Data Formats* since different LLs can share data using different formats. Also, we compare the tools based on their *Interface* offered for the users to manage the configure and manage the tools. Other criteria that we define targets the security by studying the *Data Transfer Encryption*. Also, we compare the tools based on their ability in terms of *Horizontal Scalability*, *Distribution*, *Architecture* and *Fault Tolerance*. Finally, we compare the tools based on their *Maturity* with regards to their latest stable releases and the number of active contributors in the community till August 2020.

Table I shows that Apache Sqoop is limited to ingest only structured data, e.g., data from relational databases, into HDFS systems. Furthermore, Apache Sqoop ensures communication between *Data Collection/Ingestion* layer and *Data sources* layer using JDBC connectors. However, this lacks ensuring

TABLE I
DATA COLLECTION/INGESTION TOOLS CHARACTERISTICS

| Tool | Apache Nifi | Apache Flume | Apache Sqoop |
|---|---|---|---|
| **Supported Communication** | S/FTP, TCP, DBCP, REST API, HTTP, Kafka, JMS, RPC, JDBC | S/FTP, TCP, DBCP, Exec, HTTP, JMS, Kafka, MultiportSyslogTCP, Scribe, RPC | JDBC |
| **Near-Real-time/Streaming Data Support** | Stream + batch | Stream + batch | Batch only |
| **Data Formats** | JDBC, TXT, JSON, CSV, Parquet | JDBC, TXT, JSON, CSV | JDBC |
| **Data Sources** | File systems, Spooling, Kafka, HDFS, JDBC | File systems, Spooling, Kafka, JDBC | HDFS, JDBC |
| **Transformations** | Aggregation, filtering rows, removing columns | Aggregation | No transformation |
| **Interface** | Web GUI, command line, REST API | Command line | Command line |
| **Data Destinations** | HDFS, HTTP, Solr, local files, Hive, JDBC, Kafka | HDFS | HDFS, JDBC, Hive, Hbase, Hcatalog, Accumulo |
| **Horizontal Scalability** | Scaling out linearly to at least 1,000 nodes | Supported | Limited scalability |
| **Distribution** | Supported | Supported | Not supported |
| **Architecture** | Master-less | Master-less | Centralised |
| **Fault Tolerance** | No single point of failure | No single point of failure | Single point of failure |
| **Data Transfer Encryption** | End-to-end encryption: TLS/SSL, SSH, HTTPS | End-to-end encryption: TLS/SSL, SSH | Using passwords |
| **Data Flow** | Bi-directional | Uni-directional to HDFS | Bi-directional |
| **Intended Use Cases** | Migrating data between various systems | Migrating logs, streaming event data into HDFS | Importing data from RDBMS |
| **Lineage Support** | Supported | Not supported | Not supported |
| **Maturity** | Initial release 2006 Last Stable release 1.11.4 March 2020, 267 contributors | Initial release 2012 Last Stable release 1.9.0 / January, 2019, 43 contributors | Initial release 2010 Last Stable release1.4.7 / December, 2017, 21 contributors |

security while transferring data that contain sensitive information since the security is a requirement in FEDDL. Also, for unstructured data, Apache Sqoop is not suitable since it supports only relational databases whereas FEDDL needs to ingest data available in different file formats. Instead, solutions like Apache Nifi and Apache Flume are supporting the ingestion of several file formats. Furthermore, they both offer a large choice of secure communication protocols, e.g., SFTP, which is a required while ingesting energy and customer historical data from DSOlab into FEDDL for instance. However, LLs plan to deliver historical data to FEDDL using Parquet file format which is not supported by Apache Flume. In terms of end-to-end encryption, Apache Nifi and Apache Flume offer this feature, which is very important to secure transfer of data that contain sensitive information. The above comparison helps us to opt for employing Apache Nifi to implement the *Data Collection/Ingestion* layer. We notice that Apache Nifi has the highest number of contributors. This choice is motivated also by the fact that Apache Nifi offers end-to-end encryption, which is crucial when loading sensitive data in FEDDL. Furthermore, Apache Nifi comes with many interesting built-in functionalities responding to FEDDL requirements, especially the security, i.e., end-to-end encryption, and offers support for ingesting batch and streaming data.

### *B. Data Storage*

The different file formats ingested by the Data ingestion/collection tools need to be stored in efficient ways. In the following we compare state-of-the-art distributed files systems. We exclude from this comparison Network Attached Storage systems (NAS) which are systems storing files on single machine. This can overload the network if a large number of users must be handled. Also we exclude object storage since they requires to read and write entire object to append a single line to the end of a log file for instance. In the following, we compare *Network File System* (NFS), *Hadoop Distributed File System* (HDFS), and *CephFS*.

**NFS Network File System** [11] is a protocol allowing many users to have remote access to a remote file system through the network same way as accessing a local storage on their machines. NFS was designed by Sun Microsystems.

**Apache Hadoop HDFS** [12, 13] Apache Hadoop is a distributed file system to handle large volume of data. HDFS was developed by Apache and it is written in Java.

**CephFS** [14, 15] is a distributed file system developed by Inktank Storage and it is written in C++ and Python.

For this class of tools, we consider an additional criterion, i.e., *State Handling*, to show if the tools maintain the current state and session information or not, in addition to a set of criteria already defined earlier in the paper, i.e., *Data Formats, Interface, Fault Tolerance, Architecture, Horizontal Scalability* and *Maturity*.

Table II shows that HDFS offers a large option to load and access different file formats into distributed repositories when compared to CephFS for instance where data are only acces-

TABLE II
DATA STORAGE TOOLS CHARACTERISTICS

| Tool | NFS | HDFS | CephFs |
|---|---|---|---|
| **Data Formats** | Any type of file | Any type of file | Any type of file |
| **Interface** | Command line, mount as local file system | HDFS commands, web HDFS, REST API, mount as local file system | REST API mount as local file system |
| **Horizontal Scalability** | Low capabilities | Linear scalability | Linear scalability |
| **Intended Use Cases** | Virtualization file sharing | File sharing, Data Lake, big data | File sharing, cloud servers |
| **Fault Tolerance** | Low, data is centralized | High, data is replicated | High, data is replicated |
| **State Handling** | Stateless | Stateful | Stateful |
| **Maturity** | Initial release 1984 Last Stable release NFSv4.1/ January, 2010 | Initial release 2006 Last Stable release 2.10.0/ October, 2019, 243 contributors | Initial release 2012 Last Stable release 15.2.3 May 2020, 828 contributors |

sible through REST API, or can be mounted using additional tools. The main advantage of HDFS is a mature open source solution running on commodity hardware handling a large volume of data and offering fault tolerance capabilities which respond to most of the data storage requirement of FEDDL. The competitors in the market are mainly paid solutions, e.g., storage services offered by Microsoft Azure, or MapR which is against the requirements of FEDDL since it is required that FEDDL must be built using only open source solutions. Furthermore, HDFS offers the possibility to use NFS getaway and thus users can access HDFS files the same way they access their local files, and also web interface using Hadoop WebHDFS. This flexibility helps to support uses with different skills.

### C. Data Exploration

Data stored into FEDDL has to be accessed, manipulated and queried in efficient manners. In the following we consider only data processing solutions, i.e., MapReduce initially introduced by Google and employed as their primary big data processing model, Apache Spark used for instance by Amazon, and Apache Flink adopted by Alibaba.

**Apache Hadoop MapReduce** [16, 13] MapReduce is a framework for processing large volume of data in a distributed environment initially described by Google. Apache Hadoop offers an open source rewrite of the MapReduce system and is written in Java.

**Apache Spark** [17, 18] Apache Spark is an open source framework for analyzing and processing large volume of data. It is developed by Apache and it is written in Java.textbfApache Flink [19, 20] Apache Flink is an open source framework for processing large volume of data in real-time. It is developed by Apache and written in Java and Scala.

In addition the criteria *Data Formats, Interface, Scalability, Distribution,* and *Maturity* we consider additional criteria in Table III, i.e., *Ease of Use* to study the complexity of using each of the tools, *SQL Querying Support* to show if the studied tools offers support to perform SQL queries, *Execution Level* to specify where the processing is executed, e.g., data are processed on disks or loaded into main memory, and *Workloads* to study for each of the tools which kind of workloads supported by each of the tools.

TABLE III
DATA EXPLORATION TOOLS CHARACTERISTICS

| Tool | Apache Hadoop MapReduce | Apache Spark | Apache Flink |
|---|---|---|---|
| **Data Formats** | All formats supported by Hadoop | All formats supported by Hadoop | All formats supported by Hadoop |
| **Interface** | Limited to Java programs | Command line (REPL) Java, Scala, Python, R | Command line (REPL) Java, Scala, Python |
| **Horizontal Scalability** | Linear scalability, up to 1k nodes | Linear scalability, up to 1k nodes | Linear scalability, up to 1k nodes |
| **Distribution** | Distributed processing | Distributed processing | Distributed processing |
| **Ease of Use** | Low-level code | High-level operators, Scala, Python, or Java | High-level operators, Scala, Python, or Java |
| **SQL Querying Support** | Via Apache Hive | Via Apache Spark SQL | Via Apache Calcite |
| **Execution Level** | Disk | In-memory | In-memory |
| **Workloads** | Batch Files | Batch files, interactive, micro-batches data | Stream, bulk/batch processing |
| **Maturity** | Initial release 2011 Last stable release 3.1.1/ August, 2018, 246 contributors | Initial release 2006 Last stable release 3.0.0 June 2020, 1456 contributors | Initial release 2011 Last stable release 1.10.0/ February 2020, 733 contributors |

In FEDDL, we opt to employ Apache Spark as the main framework for exploring and processing the data. Our choice is motivated by the ease of use and the speed of executing process using Apache Spark when compared to MapReduce, e.g., Apache Spark is faster up to 10-100 times than MapReduce because Apache Spark persists intermediary results in-memory rather than disks [21]. All tools offer comparable capabilities regarding scalability, fault tolerance, support of various provenance of the data, and especially their ability to process data from most types of databases and file formats which is interesting to support the evolution of FED project where new sources can be added to the project. Apache Spark fulfils FEDDL requirements since it offers native support for both batch and streaming data processing. Thus, Apache Spark responds to the future iteration of FEDDL. Furthermore, it offers a command-line interface to directly interact with the data whereas MapReduce requires writing (rather low-level) MapReduce jobs which requires expert users whereas FEDDL must provide solutions for users having different levels of expertise. Finally, Apache Spark supports both user-defined programs (in different programming languages) and SQL queries within the same framework. However, Apache Flink offers interesting features to process mainly real-time data. But, FEDDL needs to deal with both real-time and batch data.

### D. Meta Data Governance

In FEDDL, data contains personal information of customers and it is a requirement to track the different stages of the data lifecycle, e.g., ingestion, storage, or processing. Furthermore,

it is required to identify sensitive data loaded into FEDDL. Thus, meta data governance tools offer the possibility to annotate the data using custom or predefined tags during ingestion stages. Thus, sensitive data can be identified to enforce access rights while accessing data. Due to the lack of tools responding to FEDDL requirements, we limit our study to Apache Atlas as meta data governance tool.

**Apache Atlas** [22] Apache Atlas is a centralized meta data governance tool on Hadoop clusters initially developed by Hortonworks and it is written in Java.

For this class of tools we consider in Table IV the criteria, *Data Source, Interface, Horizontal Scalability, Architecture, Fault Tolerance,* and *Maturity*.

TABLE IV
META DATA GOVERNANCE TOOL CHARACTERISTICS

| Requirement | Apache Atlas |
|---|---|
| **Data Source** | Hbase, Hive, Sqoop, Storm, Kafka, HDFS |
| **Interface** | Command line, REST API, Web GUI |
| **Horizontal Scalability** | Scalable |
| **Architecture** | Centralized |
| **Fault tolerance** | Automated failover |
| **Maturity** | Initial release 2015 Last Stable release 2.0.0/ May,2019, 87 contributors |

In FEDDL, we employ Apache Atlas since it is the mainstream open source solution for governing meta data in Hadoop environments. Apache Falcon [23] could also have been considered, but it has been retired by Apache and there is thus no active development going on anymore. Another alternative is Cloudera Navigator [24], it is a closed source tool, and it does not offer broader support for services including Kafka, or Spark, and thus it will satisfy FEDDL requirements in terms of open source solutions and support for real-time data.

### *E. Access Management Tools*

In FEDDL the data contains sensitive information. Thus, it is important to consider tools providing fine-grained authorization rules for the different component of FEDDL. Due to the limited available open source tools, we opt to study the commonly used Hadoop access management tool Apache Ranger used by ING bank for instance, and Apache Sentry.

**Apache Ranger** [25] Apache Ranger is a centralized policy management tool developed by Apache and it is written in Java.

**Apache Sentry** [26] Apache Sentry is a centralized fine-grained role-based authorization system. Developed initially by Cloudera then by Apache and it is written in Java.

In addition, the criteria *Data Sources, Architecture, Fault Tolerance,* and *Maturity* we consider additional criteria in Table V, i.e., *Permission Level* to study level to which access management tools restrict access, and *Data Access Authorization* to show how each of the tools allows access to data within a specific database.

TABLE V
ACCESS MANAGEMENT TOOLS CHARACTERISTICS

| Requirements | Apache Ranger | Apache Sentry |
|---|---|---|
| **Data Sources** | Hadoop HDFS, Hive, HBase, Storm, Solr, Kafka, NiFi, YARN | Impala, Hive, HDFS, Hive, Solr |
| **Horizontal Scalablilty** | Scalable | Limited Scalability |
| **Architecture** | Centralized administration | Centralized administration |
| **Fault tolerance** | Automated failover | Limited |
| **Interface** | Command line, REST API, Web GUI | Command line, REST API, Web GUI |
| **Permission Level** | Group and User | Group only |
| **Data Access Authorizations** | Apache Atlas tags, Ranger Authorization | Apache Sentry Authorizations |
| **Maturity** | Initial release 2015 Last Stable release 2.0.0/May, 2019, 87 contributors | Initial release 2012 Last Stable release 2.1.0/ October, 2018, 27 contributors |

The comparison in Table V helps us to opt for employing Apache Ranger because it supports more applications than Apache Sentry. Therefore, the centralized administration architecture helps to manage access for several applications rather than implementing different access management policies for each of the tools composing FEDDL. Furthermore, Apache Ranger grants privileges at both group and user levels whereas Apache Sentry is limited to grant permission to the only group level. In FEDDL, users are divided into groups, e.g., LLs, and each source is represented as a user belonging to the LL group. Also, Apache Ranger offers a data access authorization system, based on tags possibly defined by meta data governance tools, e.g., Apache Atlas, whereas Apache Sentry authorization should be explicitly defined.

### *F. Resource Management*

In FEDDL, it is important to ensure resources sharing between the different components since FEDDL will be deployed on shared infrastructures offered by centre Denmark. Resource managers are responsible for allocating the available cluster resources to facilitate the distribution of running process in the cluster. In the following, we compare the two resources management tools Apache Hadoop YARN, which is widely used in Hadoop clusters, and Apache Mesos, which is used by Airbnb for instance.

**Apache Hadoop YARN** [27] Apache Hadoop YARN is one of the main components of Hadoop open source systems to ensure resources management mainly on Hadoop clusters. It is developed by Apache and it is written in Java.

**Apache Mesos** [28] Apache Mesos is a cluster manager that simplifies running applications on a scalable cluster of servers. It is developed by Apache and it is written in C++.

For this class of tools we introduce in Table VI the criterion *Scheduler* to study the level of scheduling workloads, in addition to the criteria, *Horizontal Scalability, Interface, Architecture, Supported Systems,* and *Maturity*.

TABLE VI
RESOURCE MANAGEMENT TOOLS CHARACTERISTICS

| Requirements | Apache Mesos | Apache YARN |
|---|---|---|
| **Horizontal Scalability** | Up to thousands of nodes | Up to thousands of nodes |
| **Interface** | Web GUI, Command line | Web GUI, Command line |
| **Architecture** | Master-slave | Master-slave |
| **Scheduler** | OS level scheduler | Application level scheduler |
| **Supported Systems** | Spark, Cassandra, MongoDB, Hadoop, etc. | Hadoop, Spark |
| **Maturity** | Initial release 2009 Last Stable release 1.9.0 / September, 2019, 297 contributors | Initial release 2006 Last Stable release 3.3.0 / July, 2020, 246 contributors |

The architecture of FEDDL relies on running a Hadoop cluster and thus it is natural to opt for using YARN as an underlying resources management tool. This choice facilitates the deployment task and reduces the complexity of running and maintaining the different components of FEDDL which are compatible with YARN. For instance, running Apache Spark on YARN allows Spark queries to be executed without the need for asking further permissions from the underlying resources management tools since the data are stored in HDFS and YARN is the native scheduler.

## VI. IMPLEMENTATION AND DEPLOYMENT

Figure 3 shows the set of tools selected to implement the layers in the FEDDL technical architecture. All selected tools feature long deployments in production systems and have stable releases and active communities providing support, upgrades, and bug fixes. Furthermore, we have ensured that tools from different layers can be easily integrated without advanced customization or code changes. FEDDL collects the source batch and streaming data delivered in different file formats using Apache Nifi. This ensures that data are end-to-end encrypted so sensitive data can be securely transferred and loaded into FEDDL. The large volume of data is stored using Apache Hadoop Distributed File System (HDFS). Furthermore, the wide variety of energy-related metadata is captured and managed by the Apache Atlas metadata management and governance tools. To comply with GDPR, access management is enforced in FEDDL using Apache Ranger. Only authorized partners are granted full access to the data, while Data Consumers such as EnergyTech companies, only have access to anonymized and aggregated data. The FEDDL technical architecture offers several options for partners to run their flexibility tools and solutions depeding on their level of expertise. For instance, FEDDL allows partners to execute advanced analytical jobs using Apache Spark, download data using Apache Hadoop WebHDFS, or run SQL or Python code directly on FEDDL. The latter uses web-based notebooks in Apache Zeppelin which is an open source web-based notebook tool that enables interactive data analysis.

We deploy the tools in the FEDDL technical architecture on the Center Denmark physical hardware infrastructure. We use Virtual Machines (VMs) to completely isolate the FEDDL data from the other co-located projects on Center Denmark, another GDPR requirement. To enhance performance, we dedicate physical disks to the FEDDL VMs. To enable powerful and flexible configuration management and application deployment, we use Apache Ansible [29], which is an open source configuration management tool. We configure Ansible *playbooks* which are Ansible's configuration, deployment, and orchestration language, to prepare each node with the necessary packages, e.g., a certain Java version, and install its services for configuration and monitoring. Apache Ansible prepares the VMs and installs Apache Ambari [30]. The latter automates the installation, configuration, and monitoring of software stacks in the Hadoop ecosystem, such as the tools we selected. For this purpose, we define Ambari *blueprints* which are declarative definitions of cluster configurations helping us to define our custom configuration for each of the selected tools, e.g., passwords, ports, or data and log repositories.

## VII. CONCLUSION AND FUTURE WORK

This paper presented the Danish National Energy Data Lake developed in the Flexible Energy Denmark (FED) project. This FEDDL will be the foundation for collecting and sharing all types of energy-related data in Denmark to allow AI and machine learning tools to combine, analyze, and predict energy data for the purpose of optimizing the flexible use of renewable energy. Specifically, the paper presented a case study of one of the FED living labs, called DSOlab, including the data in it, how and when they are collected, and what they will be used for, followed by the requirements for FEDDL. We then presented the layered technical architecture of FEDDL, which is composed of the *Data Sources*, *Data Collection/Integration*, *Data Storage*, *Data Exploration*, *Data Consumers*, *Meta Data Governance*, *Resources Management*, *Access Management*, and *Privacy and Anonymization (GDPR)* layers. For each of the layers, the paper compared state-of-the-art open source tools and identified the best ones based on a set of given layer-specific criteria. The selected tools fulfill both functional and non-functional requirements of FEDDL. The layered architecture of FEDDL supports the varying use of FEDDL by different groups of users such as Living Labs, researchers, and EnergyTech companies. A main challenge in FEDDL is how to deal with sensitive data about private consumers, which means that some FEDDL users have full access while other can only access anonymized and aggregated data. Finally, we discussed the implementation and deployment of FEDDL at the on-premise server infrastructure hosted by FED partner Center Denmark.

Future work will provide better support for ingesting and analyzing real-time data to enable rapid analysis and reaction times, and better multidimensional data analytics support with data cube functionality. Furthermore, we will develop advanced anonymization tools for the Privacy and Anonymization layer to balance data privacy with data utility.

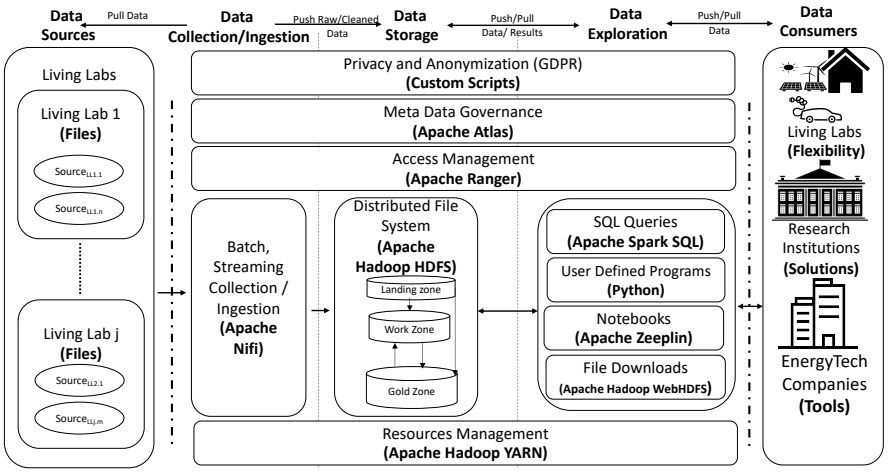


Fig. 3. FEDDL tools

## REFERENCES


[1] T. B. Pedersen, L. Šikšnys, and B. Neupane, "Modeling and managing energy flexibility using flexoffers," in *SmartGridComm*, 2018, pp. 1–7.

[2] C. J. Date and H. Darwen, *A guide to SQL standard*. Addison-Wesley Reading, 1993, vol. 3.

[3] P. Voigt and A. Von dem Bussche, "The eu general data protection regulation (gdpr)," 2017.

[4] R. Hai, S. Geisler, and C. Quix, "Constance: An intelligent data lake system," in *SIGMOD*, 2016, pp. 2097–2100.

[5] A. Gorelik, *The enterprise big data lake: Delivering the promise of big data and data science*. O'Reilly Media, 2019.

[6] "Apache Nifi," https://nifi.apache.org/, 2020.

[7] S. Hoffman, *Apache Flume: distributed log collection for Hadoop*. Packt Publishing Ltd, 2013.

[8] "Apache Flume," https://flume.apache.org/, 2020.

[9] K. Ting and J. J. Cecho, *Apache sqoop cookbook: Unlocking hadoop for your relational database*. " O'Reilly Media, Inc.", 2013.

[10] "Apache Sqoop," https://sqoop.apache.org/, 2020.

[11] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck, "Rfc3530: Network file system (nfs) version 4 protocol," 2003.

[12] D. Borthakur *et al.*, "Hdfs architecture guide," *Hadoop Apache Project*, vol. 53, no. 1-13, p. 2, 2008.

[13] "Apache hadoop," https://hadoop.apache.org/, 2020.

[14] S. A. Weil and B. et al, "Ceph: A scalable, high-performance distributed file system," in *OSDI*, 2006, pp. 307–320.

[15] "CephFS," https://docs.ceph.com/docs/master/cephfs/, 2020.

[16] J. Dean and S. Ghemawat, "Mapreduce: a flexible data processing tool," *Communications of the ACM*, vol. 53, no. 1, pp. 72–77, 2010.

[17] M. e. a. Zaharia, "Apache spark: a unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[18] "Apache Spark," https://spark.apache.org/, 2020.

[19] P. e. a. Carbone, "Apache flink: Stream and batch processing in a single engine," *TCDE*, vol. 36, no. 4, 2015.

[20] "Apache flink," https://flink.apache.org/, 2020.

[21] J. Shi, Y. Qiu, U. F. Minhas, L. Jiao, C. Wang, B. Reinwald, and F. Özcan, "Clash of the titans: Mapreduce vs. spark for large scale data analytics," *VLDB*, vol. 8, no. 13, pp. 2110–2121, 2015.

[22] "Apache Atlas," http://atlas.apache.org/, 2020.

[23] "Apache Falcon," https://falcon.apache.org, 2020.

[24] B. Quinto, "Big data governance and management," in *Next-Generation Big Data*. Springer, 2018, pp. 495–506.

[25] "Apache Ranger," http://ranger.apache.org/, 2020.

[26] "Apache Sentry," http://sentry.apache.org/, 2020.

[27] V. K. Vavilapalli and Murthy, "Apache hadoop yarn: Yet another resource negotiator," in *SoCC*, 2013, pp. 1–16.

[28] "Apache Mesos," https://mesos.apache.org/, 2020.

[29] "Ansible," https://www.ansible.com, 2020.

[30] "Apache Ambari," https://ambari.apache.org/, 2020.